HOT Z-2068

USER'S NOTES

SINWARE
Box 8032
Santa Fe, NM 87504

HOT Z-2Ø68 COMMAND LIST -- READ MODE

Command Key	Function	Routine
<=	Goto BASIC (Sign off.)	SOFF F62E
<>	Scroll display (BREAK to stop)	SKRL D9DC
>=	Turn on hex edit mode	EDMD D7C5
AT	Display machine stack pointer (switch)	SPON DADE
BRIGHT	Set border color (Ø-7)	BORS EØ7E
CODE	Switch floating-point interpretation	FPSW DAE7
ENTER	Page flip	
INK	Set ink color (Ø-7)	INKS CE36
INT	Restart HOT Z (Reinitialize)	STAR CFØØ
LN	COPY screen to 2Ø4Ø	PRSC D46F
OR	Decimal address to follow	GDEC D2CØ
OVER	Switch NAME files	SWNA D453
PAPER	Set paper color (Ø-7)	PAPS CE22
PEEK	Switch floating-point interpreter in/out	SWFP DA36
RND	Display beginning of NAME list	TOPN D3F5
STEP	Go to single stepper	VRVA DCE4
STOP	Turn on assembly mode	ASED F53Ø
THEN	Switch disassembly/data displays	DSWI E9AE
TO	Set END address	SEND F646

HOT Z-2Ø68 COMMAND LIST -- SINGLE-STEP MODE

<=	Return to READ mode	
AND	Display breakpoints	SHBP DB32
AT	Set Breakpoint #1	SBP1 DB5D
EDIT	Back one instruction (or byte if repeated)	
ENTER	Run one instruction	
INT	Run CALL or RST 1Ø	RCAL DC81
LN	COPY to 2Ø4Ø printer	PRSC D46F
OR	Set Breakpoint #2	SBP2 DB69
SPACE	Skip next instruction	
THEN	Run to breakpoint	RTBP DAFD
VAL	Set register (A,B,D,F,H,S,X,Y)	OSRS DBA3

HOT Z-2068 COMMAND LIST -- EDIT MODE

Command Key	Function	Routine
;	Escape during assembly edit	
>=	Cursor to hex edit column	SWTE D65A
ABS	Find next matching byte sequence	FIAG D356
AT	Part screen (enter address)	PSCR C1D0
CHR$	Readdress NAME file (displacement)	RANA CE00
COS	LOAD ZX81 data tape, cursor to END	LD81 C017
ENTER	Escape during hex edit, or Return to READ mode from home column	
EXP	Delete NAME	DENA D4D6
FN	Fill memory with keycode	FLMM D5B1
INKEY$	NAME entry (disassembly or data)	NENT D24B
INT	RUN from cursor to first RET	RUNT D447
LEN	Checksum to BCDE in single step	CSUM F717
LLIST	List cursor to END on 2040 printer	DLIS D404
MERGE	Transfer code and labels to DEST	TRNA CF4D
MOVE	Relocate code, cursor to END (Set TEMs)	RELO D060
READ	Hex arithmetic (E + K & E - K)	HARI D214
RESTORE	SAVE cursor to END in DATA format	SV68 D131
RND	Transfer cursor-END to DEST	TRAN CF47
SGN	Find first matching byte sequence	MATS D364
STEP	Single-step instruction at cursor	OSCO DCE0
STOP	Move cursor to assembly-edit column	SWAS F42C
STR$	Readdress jump table (displacement)	RADD D03D
THEN	Switch display (disassembly/data)	SWDD D527
TO	Set END	SEOP D776
VAL	LOAD (DATA) from cursor to END	LD68 D141
VERIFY	Verify a code-format tape	VERI D1A2

THIS IS HOT Z-2068

HOT Z-2068 combines a line-by-line assembler, a labelling disassembler, a single-stepper and a simple editor. The purpose of HOT Z is to give you a reasonable degree of direct control of your computer, as well as to assist you in writing assembly-language programs to extend your control.

The enclosed tape holds two copies of HOT Z-2068 along with a fairly full NAME file that annotates most of HOT Z and a large portion of the ROM. We have opted for a simple loading format that leaves HOT Z completely open and in your hands as a standard CODE-format tape. You should therefore be able to move it to wafer drive, disk, PROM, or whatever goodies might appear beyond the grave of TS computerdom. For EPROMs, only the code above C000 should be burned in. Variables are initialized automatically, but NAME files would have to be tape-loaded according to taste.

HOT Z-2068 is an adaptation of HOT Z-II for the TS 2068. The command keys are different from the TS-1000 version, and there are a few extra commands, but those of you who are familiar with HOT Z-II should be able to run the program with little more than the command lists at hand. HOT Z-2068 occupies memory from C000 49152 to F7C0 63424, which would be compatible with use of both display files if 64-column or extended-color software becomes available. In the absence of such systems, RAM above F7C0 may be used as workspace.

The two sides of the tape hold the same code. Use LOAD or LOAD "HOT Z" to start. The load is double, first a short BASIC routine that can be used to reSAVE HOT Z and which automatically proceeds to LOAD the HOT Z code. HOT Z comes up running, but you can switch to BASIC with the <= command and work out the details of loading or saving it from the short BASIC routine. The BASIC is for loading convenience only and is in no way necessary for running HOT Z. RAMTOP is set so that you can use NEW in BASIC without destroying HOT Z. You can change the setting of RAMTOP by just writing to that variable with HOT Z.

The large NAME file that follows HOT Z on the tape can be loaded as follows: first turn off the existing NAMEs with the OVER command in READ mode. Then set the edit cursor at address A4F2 and use the TO command to set the END variable to BF70. Then press VAL (the LOAD command) and enter NAMES in response to the tape name cursor. Hit ENTER and start the tape. When the tape has loaded, you might want to switch to data mode (the THEN key) to make sure it looks like a NAME file. Then go to BFE2 (ALNA if the NAMEs were on) and enter F2-A4-6E-BF at the four addresses there. (Note that this

differs from the ZX versions.) Then use the OVER command again (in READ mode) to turn on the new NAME list. You will find NAMEs pertaining to the ROM in lower case, those pertaining to HOT Z in caps. We will publish a full listing of these labels if there is sufficient demand for it.

HOT Z-2068 is not so easily relocatable as HOT Z-II, since there are two sections of code (See memory map.), but those of you who have learned how to use the relocator should be able to puzzle your way through the several steps required.

HOT Z will cohabit with a BASIC program,although BASIC is a foreign language to HOT Z and must be read as data when HOT Z is in command. HOT Z-2068 sets RAMTOP to protect itself from a NEW command in BASIC. The setting occurs during initialization (STAR at CF00) and can be altered to suit your needs.

HOT Z requires some knowledge of the hexadecimal (hex) number system, which uses the characters 0-9 and A-F as its 16 digits. These instructions were written with the assumption that you know the fundamentals of Z80 machine code, for which there are numerous books on the market. If you are learning, then use HOT Z as a blackboard to work out the exercises.

MEMORY MAP

A brief memory map of HOT Z-2068 is as follows:

```
BC1A     BF00          C000        C2A6       C930         CC04      F7C0
+--------+-------------+-----------+----------+------------+---------+
NAME File HZ Variables HZ Program Data Files Jump Tables HZ Program
                              (1)                                (2)
```

The command files begin with the single-stepper at CA7E, READ mode at CB00, and EDIT modes at CB82. In each case the first address is for the RND key. Other keys follow in the order they are listed in your manual.

RUNNING HOT Z-2068

The following section provides an introductory tour of HOT Z. The experienced and the adventurous among you will want to plunge right in. If so, arm yourself with the short command lists and the keyboard map and try your luck. Details of the various commands are available in the later sections of these notes.

AN INTRODUCTORY TOUR

The cover occupies the initial display file and evaporates when you press a key. Then you should see the first screen "page" of disassembled ROM. Down the left side of the screen, you will see the memory-address column, to which everything in HOT Z is keyed. These addresses are in hexadecimal and in the format accepted as input by the program. In other words, all addresses are four hex digits and include leading zeroes but no identifying symbols either before or after. The format is always there for you to consult as you make entries to HOT Z. Addresses run from 0000 to FFFF.

The second column of the disassembly display lists the contents of each memory byte, again in hexadecimal, two digits per byte, packed together with no spaces between. These numbers occur strictly in the order they occur in memory, which is not necessarily an easy order for reading. This column is raw data, as it were, against which any "interpretation" can be checked. Z80 instructions can be from one to four bytes in length. A HOT Z routine gets the length of any instruction and parses the bytes into instruction-length clusters, but it cannot decide whether those bytes hold true Z80 code, as here, or simply numbers used as data. That decision in the end is up to the reader. On this first page of ROM, the first two instructions are one byte long, the third three, etc.

The next column, the NAME column, will hold user-entered labels for the corresponding address, along with a few labels provided in a permanent file on your original tape. After you have annotated a program with these labels, you can SAVE a NAME file separately from HOT Z, to be loaded again with whatever program the labels pertain to.

The fourth column presents those particles of electronic poetry known as assembly mnemonics. Relative jumps (JR's) are listed, as in the sixth line, with their destination address (or NAME) rather than the single displacement byte with which they are coded. System variables for the ROM are listed by an abbreviated name, as in lines 4 and 5.

The first four instructions turn off the keyboard interrupt, set A to zero load DE to count 64K of memory, and jump to the initialization routine. The rest of the screen is taken up by RST routines. RST 10 prints the character whose code is in A, RST 08 handles BASIC error reports, RST 18 and 20 help with interpreting BASIC, and RST 28 is the entry to floating-point operations, which are a separate sub-language in the 2068. RST 08 and 28 are always followed by one or more (for 28) bytes that serve as data rather than as machine code. The meaning of such bytes is listed in the mnemonics column if you have the floating-point interpreter switched on.

The current HOT Z display is referred to in these notes as READ mode or disassembly. The commands in this mode are mainly for moving the display around to give access to different parts of memory. The page flip, for example, is the ENTER key; hit it to continue the disassembly with the instruction following the one at the bottom of the screen. For distant moves, you can enter a four-digit hex address to the ADDR cursor at the upper-left screen corner. For example, try ØD31 to see the initialization routine.

During address entry, you can backspace to correct an error by using the DELETE key, which will back up the cursor one space. DELETE doesn't blank out the entry and that you can't back out of the whole entry routine that way. To back out, use the ENTER key, which works as an escape key in this situation. You must type in all four hex digits of an address or all four characters of a NAME (label). ENTER is not needed after the last hex address digit.

The keyboard with HOT Z-2Ø68 responds almost identically to way it responds in BASIC. HOT Z gives a different tone feedback (You can alter that by changing pip_.) and gives the tone for CAPS LOCK and the SYMBOL-SHIFT/CAPS-SHIFT combination as well. CAPS LOCK is initially set. Lower-case a through f are not recognized as hex digits, so it you shift to lower case to enter a label, be sure to shift back before entering hex or Z8Ø mnemonics. The lower-case mode is indicated by cursor flashing and bright rather than just flashing. All the shift-key entry combinations are the same as in BASIC, except that the K-cursor state is not used by HOT Z, so the keyword legends on the keys themselves are not available.

In READ mode, you can also get to a named routine by entering the four letters of an assigned NAME. Try KEYB. You will see that the NAMEs appear in both the NAME column (referring to the current address) and in the mnemonics column (referring to the target address of CALLs or jumps).

In general, you can use a NAME in the file as a proxy for its address in the READ, Assembly-Edit, or One-Step modes of operation.

Try keying THEN from READ mode. This is the display switch, and successive strokes of the the same key will take you back and forth between the data and the disassembly displays. The data display is for examining those parts of memory that are used as files of data rather than for Z8Ø code. The first and second columns contain the single address and its content in hex, values that are reflected in decimal in columns four and five. (Use it as a conversion table.) The far-right column gives the CHR$ of the contents of the address and will turn up any BASIC programming or message files. Enter, for example, the address Ø227 to see the keyboard file. Switch back to disassembly while you're still looking at the keyboard file

for a taste of what disassembled data (sometimes called nonsense) looks like. It's up to you to distinguish sense from nonsense when reading a strange program; the display switch is there to help you do it.

The NAME column in the data display functions differently from the column with the same heading in the disassembly. The NAMEs in the data display are those that correspond to any two successive bytes, taken in lo-hi order, in the second column. (The disassembly displays NAMEs assigned to the addresses in the first column.) Some NAMEs in the data display can crop up by chance; for example, two NAMEs immediately together mean that at least one is spurious.

Use the RND command in READ mode to go to the beginning of the NAME file. The NAME file grows downward like a stack, which it is not, as you add new NAMEs to memory addresses. Turn on the data display to see the structure of the NAME file. Each NAME takes six bytes; the first two hold the address to which the NAME is assigned, hence the listing in the NAME column, and the next four hold the NAME itself, which shows in the CHR$ column. Other odd CHR$ symbols will appear at random for some of the address bytes, signifying nothing.

The data display is also useful for looking at BASIC programs to see the real structure of BASIC code.

You can enter decimal addresses to the ADDR cursor, but these must be prefixed by the OR command, which will put up a D after ADDR. Try it, and check the conversion with the data display. If you enter a decimal address of less than five digits, then you have to press ENTER to tell HOT Z that you've finished. If you enter a decimal higher than 64K, the program will subtract 64K and give you what's left.

Now get into disassembly and go to 3B2E, which is where the ROM begins the BASIC function LN. Hit PEEK to turn on the floating-point interpreter. The first instruction after the RST 28 restacks the number on the top of the calculator stack in full five-byte form (in case it is a short integer); the number is then duplicated on the stack and tested for being positive non-zero; if it is, a jump is made to 3B37; otherwise, execution proceeds to end the floating-point code and fall into the trap for error A. At 3B37, we have an example of floating point code that is embedded and not preceded by an RST 28 because of the jump. To get the correct interpretation, enter 3B37 to the ADDR cursor, then use the switch command on the CODE key.

At 3B35 you will see a rendition of a BASIC error report after RST Ø8, in this case for a zero or negative argument to the logarithm. Occasionally, you will encounter a CF as data rather than RST Ø8, in which case the error number may be invalid and left blank.

The last display on the tour is the Z8Ø register display or Single-Stepper. It is one of the quirks of bilinguality that this display must be entered from an area where the floating-point interpreter is not switched on, so first enter an address above 4ØØØ, say. Then use the STEP command from the disassembly.

The register display occupies the top three quarters of the screen. The left column lists the various Z8Ø registers; please refer to a good Z8Ø reference book if you need an explanation of the register names. The exchange flags are listed as EXFLAGS.

The second column lists the hex values of the registers' contents. Values for the accumulator (A) are listed at the left of the column to remind you that A is the high half of the AF register pair, along with H, D and B. The third column either converts the second column value to signed-decimal according to the two's complement convention, or, if the second column holds an address that has been NAMEd, then that NAME is listed in the third column. The fourth column, headed by the open parentheses, gives the hex value of the byte contained in the address formed by the register-pair values. (E.g., across from HL you will find the byte (HL).) The right column gives the CHR$ of the byte in the fourth column (for the register pairs) or of the byte in A.

The box below the one containing the exchange registers holds details on the one-step user's stack and the state of the flags registers. The user's stack is separate from the main machine stack so that the system can absorb a few stack errors without crashing the program. The top four pairs of bytes on the user's stack are shown at the right, along with the NAMEs for any addresses they might hold, so that you can check to see whether your test routines leave anything behind. The main flags are listed below the exchange flags for easier visual association with the conditionals in the program steps below. Standard conditional mnemonics are given for the four programmers' bits.

The cursor at the left in line 18 (which is bright) marks the address of the next step set up to be executed by the single-stepper. You can enter any address into that cursor just as you would in READ mode, or you may also use a NAME. The ENTER key still serves as an escape during address or NAME entry, but it has another more important function as well, which is to run the next single step.

If it's not already there, enter Ø53A to the NEXT slot, and then notice the contents of the A and C registers just before and after you press the ENTER. This is a fairly safe area and you can experiment with a few more steps. (The things you must be careful about are loading into some system variables, either ROM's or HOT Z's, and some flag sets. The SPACE key allows you to skip the step at NEXT. The top line of Z8Ø instructions represents the previous step executed, and the three steps following the one in NEXT are those that will be reached if there is no branching. A branched-to step appears directly in the NEXT slot; a skipped step disappears from the display.

For faster debugging, you can set breakpoints (AT and OR commands) and use the THEN command to step through the code as far as the first breakpoint encountered. Two breakpoints are provided so that your can cover both sides of a conditional branch. You must take care to set breakpoint addresses that the code will actually encounter, since stopping depends on finding a breakpoint exactly. The BREAK key will stop the THEN command if used quickly enough. You can display the current breakpoints with the AND command.

Learners might consider mastering the use of the Single-Step first and then using it to see how the various instructions and a few resident routines work. A lot of bugs can be avoided by testing every routine you write with this device.

Hit <= (Quit) to get back to the main READ display. You will arrive at a screen page that starts with the address that was in the NEXT slot of the Single-Stepper. If you spot an error coming up at the bottom of the Single-Step display, you can quit the display, EDIT the error on the disassembly display, and get back to where you were in the Single-Step by using the shift-S command from READ mode.

Writing and Editing Z8Ø Code

The READ mode is a essentially passive, allowing you to page through the memory and examine its contents. The WRITE or EDIT modes are there to let you make changes in the memory content, provided that memory is RAM.

There are three WRITE/EDIT modes. With the disassembly display, you can press STOP and a cursor will appear at the top line of the edge of the right column. This is the assembly mode. Once you turn on the cursor, you change the entire command system of HOT Z. The commands available to you with the cursor on are listed as the EDIT-mode commands on the command lists. Hitting ENTER with the cursor in its "home" column will quit the WRITE mode and return you to READ, where you can readjust the screen to another part of memory.

In addition to the command set, the up and down cursor controls allow you to move the cursor to a given line or to scroll the display page one line up or down by moving the cursor up from its top position or down from its lowest position. Up scrolling is automatic when you ENTER a line that is third from the screen bottom.

You may also enter a new Z80 instruction to replace the one listed on the cursor line. Just start typing and the existing line will disappear. As you type, the delete key and the left and right cursor controls will function as you expect them to. If the cursor is over the top of a character, your next keystroke will replace that character. If you want to insert a character, press the EDIT key and a space will be created at the cursor position, with all characters to the right of the cursor being shifted one space right. The rightmost character in the line (usually a blank) is destroyed by this insert command. You cannot jump to another line with the up or down cursor command while you are in the middle of editing a given line.

When you have entered the intended Z80 instruction, hit the ENTER key to put the proper code into memory. If your entry is in the proper format, the cursor will return to the left edge of the column and move one line down, ready to edit the next line. If the cursor stays put in the line you are working on, then it indicates a format error in the mnemonic entry.

HOT Z-II follows the format of the mnemonics listed in the Zilog Z80 technical manual. This format is the same as that listed with the character set in your computer's instruction manual, with the following exceptions: the RST's are followed by a hex byte (08,10,18,20,28,30,38) rather than decimal and the OUT (N),A and IN A,(N) use the parentheses shown here. (N is always a two-digit hex byte.) The open parenthesis is always preceded by either a space or a comma, and spaces are always important.

When HOT Z fails to accepts your entry, it locates the line cursor at the first position that does not match its template for a proper instruction. Sometimes, however, as with an omitted space or an unassigned label, the cursor may appear earlier than your particular format error. (For example, it will flag the first letter of a label even if only the fourth letter is "wrong".)

If you get stuck and can't get HOT Z to accept what you've entered, you can abandon ship and restore the original mnemonic by hitting the semicolon (;). Your recourse then is to look elsewhere in the disassembly for the format of the instruction you have been trying to enter, or to look up the hex code for that instruction and to enter that in the hex column (See below.) to discover how HOT Z lists the mnemonic.

If you try to back out of a line with the cursor-left key, HOT Z will act as if you have tried to ENTER the line. If you write all the way to the end of the line an ENTER will also be automatically appended. This occurs with some of the IY+N instructions, which just fit in the alloted space.

You can use a preassigned NAME in an instruction anywhere that a 16-bit (four hex digits) number occurs. For example, LD HL,(rmtp) is equivalent to LD HL,(5CB2). You must give a NAME to a particular address (INKEY$ command in WRITE) before you attempt to use it in an instruction.

Relative jumps (JRs and DJNZ) are normally entered with the destination address or NAME. However, for the JRs only (not DJNZ) a second form is available for short forward jumps where you haven't yet assigned a NAME but know how far forward you want to jump. JR +5 will jump ahead over five bytes. The plus sign is required and the displacement is in *decimal* with a range from Ø to 127. Backward jumps are not catered for in this way; it is easier to look back for the address you want to get to.

Provided you do not want one of the last four conditional expressions (M, P, PO, or PE), you can use relative jumps all the time, and if the destination address is too far away HOT Z will convert your JRs to JPs (absolute jumps) rather than report an error. The reverse is not true: if you enter a very short absolute jump, HOT Z will take your word for it. This conversion works well for entry of new code, but you must beware when editing in the middle of an existing routine, because if a two-byte JR is edited and becomes a three-byte JP, then the first byte of the following instruction will be overwritten.

There is no ORG command because you are doing the ORG yourself with HOT Z. However, direct data entry is possible in the assembly-edit mode through use of the DB pseudo-op. DB may be followed by a quoted string (DB "ABCDE") or by an even number of hex digits (DB Ø9ØF ØD3A). Spaces are ignored in reading the hex digits, except for the required space after the DB. Each pair of hex digits is read as one byte, and a single digit left over will be ignored. You can write a string or series of digits all the way to the end of the line.

When you hit the end, HOT Z will add the quote if necessary and enter the line. Upon entry, the editor enters one character (for a string in quotes) or two hex digits per byte starting with the cursor address for as many bytes as it takes, then resets the screen layout so the next cursor address is at the top of the screen. The reason for this is that the data you have entered would be disassembled by HOT Z, producing a nonsensical listing. You can look back with the data display to assure yourself that what you have entered is indeed there.

The DB is simply a means of entering data without leaving the assembly-edit mode. You should still assign NAMEs to your strings or variables and use them in referencing the data. The insert command is recommended when you enter data into an existing code block.

If you want to use the RELOCATE command (described below), then you should not mingle small blocks of code and data. Keep them in large blocks and keep track of what is where.

In addition to string entry with DB, you may also enter quoted non-inverse characters for direct eight-bit register loads or for direct arithmetic/logic operations. LD A,"A" will assemble as LD A,41 and CP "Z" as CP 5A. Sixteen-bit (double) register loads are not treated in this way.

Hex Edit Modes

Hit the >= key with the disassembly display to get into the main hex edit mode. The "home" column for the cursor in this case is between the address and hexcode columns at the left of your screen. Cursor controls work as with the assembly-language editor.

To change the hex content of memory, you may either move the cursor over with the cursor-right key or retype the line, using the keys from Ø to F. With the disassembly display, each line holds the correct number of bytes for a single Z8Ø instruction. If you write a one-byte instruction, the cursor will jump to the next line immediately; for multi-byte instructions, the cursor waits on the line until the required number of bytes have been entered, then jumps automatically. The purpose of this feature is to allow you to copy hex listings from printouts or magazines. You can just type away without worrying about hitting ENTER at every line, and the screen will scroll along with your entries.

With the edit mode, what you see in the hex column is what you get when you make an entry, byte for byte. Edit does not use NAMEs and you have to calculate the displacements for any relative jumps you enter.

All of the EDIT-mode commands are available with the hex-edit cursor on screen. There is, however, no character insert while you are editing a line, and the escape key in the middle of a line is ENTER rather than semicolon. If you need to change the first byte of a line after you have started editing it, you should escape by hitting ENTER and start over.

You can hit the THEN (display switch) key either before or after you have gone to the hex-edit mode in order to obtain the data-edit mode. This mode lets you change one byte at a time by writing a new value over the top. This is the mode that you would use for entering hex data files, addresses and the like. (Use the DB command from the assembly mode for entering text files.) All write commands are available from this mode as well, except the NAME (INKEY$) command functions differently than it does with the disassembly display. INKEY$ will no longer assign a new NAME, but can be used to write a preassigned NAME to the NAME column, and the address to which that NAME belongs will then appear at the cursor address and the byte following. The intended use is for creating address files (jump tables).

Inserting and Deleting Lines (All WRITE/Edit Modes)

What happens when you press ENTER after writing an instruction is that HOT Z reads the address of the line you are working on, looks up the the numeric code of the instruction, and enters that code into as many bytes as it takes. Then control goes back to the disassembler, which reads back your code into Z80 mnemonics and revises the screen page accordingly. An important consequence of this is that when you are editing an existing block of code you must be careful not to overwrite more lines than you intend to (by entering a four-byte instruction over a two-byte instruction, say) and to watch out for new instructions that crop up when you overwrite a long instruction with a short one (one-byte over a three-byte instruction, for example).

If you don't know the byte length of Z80 instructions, the way around the above problem is to use the line-insert (EDIT) and line-delete (DELETE) commands whenever you are editing an existing block of code.

When you insert or delete a line, a block of code is moved either to make room or to close up the empty space. One end of that block of code is determined by the cursor; the other end must be determined by you before you start your editing session. Whenever the WRITE cursor is on, a variable called END is displayed in the upper right corner of your screen. END marks the other end of the active memory block for an insertion or a deletion or indeed for any block operation, such as a clear, a fill, a SAVE, or a transfer. END is set with the TO key (as in TO the END) followed by four hex digits or a NAME. On some types of entry errors, you may be asked twice for the proper value.

You should set END whenever you begin an editing session. END should be within your workspace and not overlap with the HOT Z program, lest you move sections of HOT Z around and lose control of your computer. For the insert-line and delete-line commands, a special restriction has been added to the value of END. For those operations, END must be within 256 bytes of the cursor address, or else you will be asked (automatically) to enter a new value of END when you give the insert or delete command. At that point, HOT Z will accept any value you enter for END and perform the operation. The purpose of this behavior is to catch those times when you have forgotten to set END, and to save you from a possible crash.

For insertions and deletions, END can be either above or below the cursor address. The "usual" value would be for END to point to an address higher than the cursor address, in which case an insertion would push all values to higher addresses to make room for the new instruction. For example, if you insert a two-byte instruction at 8C10 with END set to 8C80, then all instructions from 8C10 will be moved two bytes higher until 8C7E, which will go into 8C80, and the original contents of 8C7F and 8C80 will be destroyed. A deletion of a two-byte instruction would move all instructions to lower addresses, and the contents of 8C7F and 8C80 would be duplicated in 8C7D and 8C7E.

On the other hand, if the address in END is lower than the cursor address, then an insertion will leave the following addresses undisturbed but will push the contents of preceding addresses to lower addresses as far as END. For example, with END set to 8C00 and the cursor at 8C10, insertion of a three-byte instruction would destroy the contents of 8C00, 8C01 and 8C02 by overwriting them with the contents of 8C03, 8C04 and 8C05, respectively. Analogously, a deletion would duplicate the first three (or N) bytes in the next three. The insertion itself will in this case go into the address *preceding* the cursor address. This feature is useful when you are editing in a constricted memory block with blanks that may be either above or below.

After insertions or deletions, the cursor position may have to be adjusted for your next entry. (The preceding discussion uses "above" and "below" to refer to numerical values of addresses, not to screen position, where addresses get higher as you go down the screen.)

When a NAME is assigned within a block where you are inserting or deleting lines, the NAME will move with the instruction to which it is assigned. The displacement assigned to relative jumps is not adjusted, so JR TARG may read JR 8C22 after an insertion that pushes TARG from 8C22 to 8C23. Be sure and label all JR destinations and then check that the labels are still correct after an editing session. If you use labels all the time, then an error will stand out clearly.

When you are editing the data display, all insertions and deletions affect one byte at a time.

Using WRITE Commands

Many of the WRITE commands affect a block of memory and require that the END variable be set first to a proper value. Use the TO key to set it. Aside from its use for insertions and deletions of lines, END is generally set to denote the end of a block of code, whereas the cursor marks the beginning. If END is less than the cursor address, the block is generally taken to be null, though sometimes the operation will still affect the very first byte. Most operations include the END address; the exceptions are SAVE and LOAD, which finish one byte before. (This makes it effectively impossible to LOAD or SAVE address FFFFH, since the next address is 0000, which is less than any cursor address.)

The block commands are LOAD, SAVE, FIND, transfer, clear, fill, print, readdress and relocate, in addition to the line insert and delete described above. The simpler commands are STOP and >=, which toggle the cursor across the screen between assembly-edit and hex-edit; THEN, which toggles the display between disassembly and data and works only in hex-edit because you can't assemble data; INKEY$ and EXP, which allow you to assign or delete a NAME at the cursor address; STEP, which takes you to the single stepper; and INT, which transfers control to the program beginning at the cursor (Novices beware!); and the PAPER/INK/BORDER color commands, which take a number from 0 to 7 after the command key.

The cassette commands (LOAD, SAVE, VERIFY) allow you to move the contents of individual blocks of memory to and from tape in the CODE format. Such tapes will be loadable by the corresponding BASIC command if you calculate the length (END - cursor address) and work out the decimal values. Similarly, CODE-format tapes made in BASIC will load with HOT Z when you have made the numeric conversions to hexadecimal. The BREAK key works to interrupt any of the cassette functions. Error reports will appear on screen with a BEEP, and the system will wait for a keystroke before accepting any further commands.

Cassette functions all require tape names, which are entered without quotes after you give the command and before you press ENTER. Maximum length for such tape names is the standard 10 characters.

The TRANSFER command allows you to move the contents of one block of memory to another block. The first thing to do is to make sure that your destination block will hold the source block without overwriting something you want to keep (or HOT Z). You have the option of copying just the code (RND) or of copying the code and moving the NAMEs assigned to it as well (MERGE). The original of the code will not be erased by this command. You can copy from ROM but of course not into it.

To use the transfer command, set END and hit the appropriate command keys. This will bring up a DEST cursor at the upper left, which asks you for the destination address of the block. HOT Z will wait for you to hit ENTER after that address, and if you change your mind or find you've entered it incorrectly you can bail out by hitting the SPACE key instead of ENTER. After the command has executed, the display will move to the address you gave to DEST.

The FIND command has a similar protocol to that of transfer. In this case, set the cursor to the beginning of a block of memory for which you want to find a match. Set END to the last byte of your template. Hit SGN. An address cursor labelled LOOK will come up at the upper left. Enter the address at which the search should begin; hit ENTER to proceed or SPACE to back out. HOT Z will search 32K (8000H) bytes for a match to the memory from cursor to END; if a match is found, the display moves to it; if there is no match, the display remains at your template in READ mode. If you find one match and want to search for another, set the cursor again, move the cursor down a line or two so it doesn't point to the beginning of the found match, and use the ABS command. If a second match is found, the display will move to it; if not, the display stays put. (NOTE: If you are searching for a block of 8 zeroes, say, and you find a block of 12, then to continue the search you should move the cursor down so that there are 7 zeroes or less below it, or else you will find the same string all over again.

The CLEAR command (ERASE) will put zeroes in all bytes from cursor to END. The FILL command will first ask you for a keystroke and then fill the block with the code for the character assigned to that key. (If you clear or fill a block of HOT Z or the stack, you are likely to crash.) To fill with a value not available from the keyboard, write that value to the HOT Z variable FILC, then use the CLEAR (not FILL) command.

The LIST command in WRITE will send the contents of the screen, starting with the cursor line, to your 2040 printer. Printing will continue, interrupted by page flips of the display, until the line just before the END address. If you forget to set END, you can BREAK to save paper.

There is also a hex-arithmetic command, which, though not a block command, uses both the cursor address and END. The command is READ, and the result is the hex sum and difference (END minus cursor address) of the two values, which are displayed in the command (top) line.

The Readdress (for jump tables and NAME files) and Relocate (for programs) commands are described in a later section of these notes, due to their complexity.

A detailed description of all the HOT Z commands is also included as a later section intended for occasional reference. For normal use, you may want to detach the brief command lists included at the beginning of these notes.

Other sections will give you details on naming and NAME files, the floating-point language interpreter, and the program relocator. If there are specific commands which you find absolutely opaque or unusable, please write to us for details.

HOT Z's Flags

HOT Z uses the BASIC system variable STRLEN as 16 bit-flags, so you could crash the system if you try to load that variable. The significance of the bits at 5C73 is as follows:

0	Set for disassembly of RST 08h
1	Set for disassembly of RST 28h
2	Set for an INSERT in progress
3	Set by an input NAME, reset by an ADDR
4	Set for data display
5	Set for EDIT, reset for WRITE
6	Set for a scroll
7	Set for window in register display

This use does not, to our knowledge, affect the operation of a co-resident BASIC program.

DISASSEMBLER FEATURES

The HOT Z disassembler has been specially programmed for the Sinclair ROM to take account of the system variables, the BASIC error reports, and the floating- point operations, which make up the Sinclair 'calculator language'.

Abbreviations of system variable names are included in the permanent NAME file that loads with the program. The HOT Z disassembler always uses the name for a system variable whether it is referred to by absolute address (e.g. 5C72) or by a displacement from IY (IY+38). However, if you want the IY form from the assembler, you must write it out, since the assembler will always substitute an address (two bytes) for an entered NAME. Since the system variable names are part of a NAME file, you can change the abbreviations to suit your own taste by entering a new NAME over the top of the old one (INKEY$ command in WRITE).

When an RST Ø8 occurs, the following byte is not Z8Ø code but is used as data to generate the BASIC error report. HOT Z reads these bytes as ERROR 9, etc., rather than generating Z8Ø mnemonics for them. If you are running the disassembler over a block of data, you may encounter a CF (hex for RST Ø8) followed by a byte that would be out of the range of the error reports. In that case, the error number is not printed.

An RST 28 is the ZX ROM's entry into the floating-point language, which can be disassembled by HOT Z. If you find this second language distracting, you can switch off the f-p language interpreter with the PEEK command (READ). If you want to know what is going on in the floating-point routines, then consult appendix A of these notes.

NEW COMMANDS ON THE 2068

The following commands were not on HOT Z-II:

READ MODE

<>	Scrolls the display for a quick look through memory. Turns off the floating-point interpreter. Stop the scroll with a BREAK.
BRIGHT INK PAPER	HOT Z-2068 is better in color due to the use of bright highlighting for cursors. These three commands set BORDER, INK and PAPER respectively. Following the command, you will be asked for a color number, which must be in the range 0-7.
TO	Set END. Allows the END variable to be set without going into an edit mode.

WRITE MODE

AT	Parts the screen display to allow you to have one small block of addresses at the top of the screen and a non-contiguous block at the bottom. The command moves the cursor to the far left of the screen. Write the desired address there and a new disassembly will fill the screen below.
CHR$	Readdresses a label file by a displacement to match NAMEs to relocated code. Works in the same way as the command that readdresses jump tables.
COS	Loads ZX81/TS1000 tapes as data from cursor to END. Works in nameless mode only and loads the first band that comes up.
LEN	Checksum from cursor to END. Gives a 32-bit checksum in BC (hi) and DE (low) as values in the single-step display. From 0000 to 3FFF on our machine gives a sum of 001ABC81.
VERIFY	Verifies a CODE-format tape from cursor to END. An error report for a bad tape will wait for a keystroke to resume HOT Z.

THE COMMAND SET

We have attempted to maintain some consistency with earlier versions of HOT Z in the choice of command keys. Mnemonic associations are generally with the letter on the key: for example, assembly is Symbol-Shift/A, the STOP key. Remember that you can reassign any command to any key by moving addresses in the command file.

READ Mode

Key	Description
STOP	Sets the cursor to the top line and switches to the assembly-edit mode. The same keystrokes will get you from hex-edit to assembly edit. This command works only when the disassembly display is on.
AT	Switches on or off a display of the stack-pointer address in the upper right screen corner. The default is Off, because it isn't pretty, but you should turn it on when you are test running your own routines. There is a small amount of shock absorption in the HOT Z stack, but if you should see it changing, then look very carefully at what you are doing to the stack with the routine you are testing. Restarting HOT Z will reset the stack.
THEN	The display switch from disassembly to data display or back again. The same command works with the hex-edit cursor on but not from assembly-edit.
>=	Sets the cursor to the top line and switches to the hex-edit mode. This command also works from assembly-edit mode without resetting the cursor line.
OVER	NAME file switch. If you are using only one file, the NAMEs are switched off or on. If you have two files in memory, the command will switch from one file to the other. The point of the double NAME file is for revising a program under development, so that you can use the same NAMEs at two different addresses.
<=	Quit HOT Z for BASIC. HOT Z remains resident and can be recalled with RAND USR 52992.

INT — Restarts HOT Z. Reinitializes variables and resets the stack to clear clutter.

STEP — Switch to single-stepper. The address in the NEXT and LAST slots will be last ones used there. Use this command to get back after you have spotted and repaired an error in the upcoming code. All old single-step register values are preserved.

RND — Move the display to the start of the NAME file and switch to the data display. Use this command as preparation for SAVing a NAME file. (Turn on the cursor, set END, and SAVE.)

OR — Indicates decimal address to follow. The command will add another inverse block to the ADDR cursor. If the decimal address is less than five digits long, hit ENTER after the last.

CODE — Floating-point interpreter switch. This is a flag switch (NOT an on-off switch) which switches interpretation of a byte from Z80 language to floating-point language. This command is necessary for certain embedded sections of floating-point code that are not preceded by an RST 28 but are jumped to from some other portion of floating-point code. This command will not function if the PEEK switch has been set to off. If it doesn't work, hit PEEK and try again.

PEEK — Switch the on-off state of the floating-point disassembler. If turned off, then the CODE command will have no effect. If on, then every EF (RST 28) will switch to the floating-point disassembly and every 38H will switch off the floating-point disassembly. If you have a stray EF on screen while you are in an edit mode, you may get a messed up display when you enter code. If so, exit (ENTER) from edit mode, use this command, and go back into the active mode without fear. Default state is OFF.

LN — Copies the screen. Useful for small routines. Gives you headings and all. Consider using the LLIST command from an edit mode for no headings and variable length.

WRITE Mode Commands

STOP Switch to assembly-edit mode. Works only when disassembly display and edit mode are on. Moves the cursor horizontally.

VAL LOAD from cursor to END. Loads 2068 CODE-format tapes. Set the cursor to the start address and END one byte beyond the last, such that END minus cursor address equals the byte length. Unlike the BASIC command and earlier versions of HOT Z, a tape name is always required byt this command. No quotes are used.

THEN Display switch, data/disassembly. Works only from hex-edit mode.

\>= Switch to hex-edit mode from assembly edit. Moves the cursor horizontally.

EDIT Sets the Insert mode for the next instruction (only) to be entered. If END is less than the cursor address, then instructions are pushed to lower addresses (up the screen) as far as END; if END is greater than the cursor address, then instructions are moved to higher addresses (down the screen) as far as END. Any NAMEs assigned to shifted memory area will also be shifted so that they stay with the instruction to which they were assigned. Relative jumps to or from the shifted area are not corrected and may require a fix-up. If END is 256 bytes or more from the cursor address, you will be required to confirm the END value before the operation proceeds.

ENTER Quit to READ mode when cursor is in "home" column. During hex entry, ENTER escapes and leaves the original memory contents intact. During mnemonics entry, ENTER sends the line contents to the assembler for entry into memory.

SGN Find the string marked by the cursor (first byte) and END (last byte). Sets the display to start with the found string. If no match is found, then the display remains at the template string. To find the next match without going back to the template, use ABS. Do not use other commands between SGN and ABS.

;	During mnemonics entry, escapes and leaves the original memory contents intact.
INKEY$	NAME command. This command has two separate effects, depending upon whether it is used with the disassembly display or the data display. With the disassembly display, the effect is to christen that instruction with the NAME that you enter to the screen following the command. A NAME requires four characters with at least one beyond F in the alphabet. (All of lower case works.) Space and semicolon should not be used. With the data display, the NAME you enter following the command must already be assigned to some address. HOT Z then looks up the address for that NAME and pokes that address to the byte at the cursor address and the byte following, then moves the cursor down two bytes. Use this form for entering tables of addresses.
DELETE	Deletes the instruction at the cursor and closes up the code between the cursor and END. END may be either lower or higher than the cursor address. If END is less than the cursor address, then code is moved from lower addresses to close the space; if END is greater than the cursor address, then code is moved from higher addresses to close the space. Code at the END address and beyond (moving away from the cursor) is preserved. If END is 256 or more bytes away from the cursor, then you will be asked each time to verify the END value before the command is executed. The purpose of this is to prevent your messing up the entire memory by forgetting to set END properly.
READ	Does hex arithmetic. Takes the cursor address (K) and END (E) and displays on the top line the sum (E+K) and difference (E-K) in hexadecimal.
INT	Runs code beginning at the cursor address. Returns to HOT Z with the first RET. If you do an extra POP and destroy the return address, then you are on your own. (This command differs from the similar one in HOT Z-1, which requires a JP back to HOT Z.) Recommended procedure is to test your routines first with the single-stepper before attempting the R command.
STEP	Single-steps the instruction at the cursor address and switches to the single-step display with the result of of that instruction in the register values and the following instruction in the NEXT slot.

RND Transfers code between the cursor address and END (inclusive) to a destination (DEST) that you enter following the command. ENTER after DEST executes the command; SPACE after DEST cancels the command; TO after DEST lets you reset END before the command is executed. Does not transfer NAMEs. To do that, use the MERGE command, which is otherwise identical to this one.

EXP Deletes the NAME at the cursor address from the current NAME file. This command will only affect the NAME that you see on screen with the disassembly display, so it is best not to use it with the data display.

TO Brings up the END? cursor that allows you to reset the END variable. Whenever a block of code needs to be marked, it is generally delineated by the cursor address and the address assigned to END. Always use it to block out a segment of memory for Insert and Delete commands before beginning to edit. END should be set within 256 bytes of the cursor for editing, but that restriction can be overridden in any particular case. (See Insert and Delete instructions.)

RESTORE SAVEs code from cursor to END-1. Enter a tape name without quotes. This is a CODE-format SAVE. You can reload such tapes from BASIC by converting the cursor address to decimal and setting the byte length to END minus cursor address.

Y Outputs the screen without headings from the cursor address to END to the 2040 printer.

ERASE Clears memory from cursor address to END. Take care not to erase HOT Z or your own programs.

FN Fills memory from cursor address to END with the code for a key that you specify in response to the KEY? prompt. For unkeyable values, write that value to the HOT Z variable FILC and then use the ERASE command.

MERGE Transfer memory contents and assigned NAMEs from a memory block (cursor address to END, inclusive) to an area beginning with an address entered in response to the DEST prompt. (See RND command.)

STR$ Readdress a jump table (address file) between the cursor address and END by a 16-bit displacement value entered in response to the DISP prompt. Takes the address (lo-hi order) at each pair of memory locations, adds the displacement, and re-enters the sum to the same locations.

MOVE Relocates Z80 code between the cursor address and END. Readdresses all CALLs or JPs. Allows a three-way partition of code, variables and (constant) files. Requires nine addresses to be first entered at TEM1 through TEM9. See the special instruction sheet on this command.

Initializes display window for single stepper. Set the cursor to a block of 6K bytes of clear memory and give this command. Then go to Single-Step mode and use the switch to see the result of those steps that put a character on the screen.

ABS Continues the search for the string specified in the F command. Starts searching from the current cursor position. (If, for example, you are searching for a block of six empty spaces and you find a block of nine, then you should move the cursor down four spaces or more, so you don't refind the last eight spaces, then the last seven, etc., of the same block.) Uses temporary variables that could be overwritten if you stop in between for other operations.

SINGLE-STEP MODE

Key	Function
AND	Display breakpoints. Lists the current setting of the two breakpoints on the line below the flags display.
EDIT	Backs up. On its first use, this command takes the instruction from the LAST slot at the top of the disassembly listing and puts it in the NEXT slot (second line). Repeated use with no intervening commands will back up one more byte for each keypress. Intended use is just to get the last step back.
ENTER	Runs the instruction in the NEXT slot and reports the resulting register values.
THEN	Go (run) to breakpoint. Causes the test routine to run from the address in the NEXT slot to either of the two breakpoints, which must be set in advance of this command. Breakpoints must be set to an address that starts a command and not to a byte embedded in a command. The GO routine checks the BREAK key after executing each line of code, so you can recover from endless loops and sometimes from runaway routines (if you're quick) by hitting BREAK.
<=	Quit single-step and return to READ. Return address is the address in the NEXT slot of the single stepper. Register values will be preserved if you reenter from READ mode.
INT	Run a CALL or RST 10. It is your responsibility to know that the called routine will not crash and not to send RST 10 any unprintable characters. The purpose of this command is to shorten the time needed to step through complex routines.
VAL	Set register value. The response to this command will be REG? in the NEXT cursor. You should respond as follows for the various registers:

A for the A register
B for the BC pair
D for the DE pair
F for the Flags register
H for the HL pair
S for the user's Stack Pointer
X for the IX pointer
Y for the IY pointer

(VAL) Note that all settings are 16 bits (two bytes) except for the one hex byte for A and the mnemonic setting for F. The specific flag bits are set or reset with the same mnemonics as are reported (M, P, Z, NZ, PO, PE, C, NC). Use this command to set up initial conditions for testing your routines.

SPACE Skip the step in the NEXT slot and advance to the next instruction. Skipped instructions are not listed in the LAST slot at the top of the disassembly segment.

Turns off a window so that you can reclaim the memory space devoted to it. To use a window again after this command, you must begin again with

AT Set Breakpoint2. Breakpoints are set just as register pairs are, with a NAME or address entry into the NEXT cursor. You must set the breakpoints precisely to the beginning of the instruction at which you want the single-step to stop, because the stop depends on the address of the next step being exactly equal to the breakpoint. If the breakpoint points to the second byte of a two-or-three-byte instruction, you routine will never stop until you crash or hit BREAK.

OR Set Breakpoint1. Breakpoints are set just as register pairs are, with a NAME or address entry into the NEXT cursor. You must set the breakpoints precisely to the beginning of the instruction at which you want the single-step to stop, because the stop depends on the address of the next step being exactly equal to the breakpoint. If the breakpoint points to the second byte of a two-or-three-byte instruction, you routine will never stop until you crash or hit BREAK.

Window switch. Switches the optional full-screen display after each step. The first time you hit shift-W switches the display in, the second time switches it out, etc. Before you use this command, you must first have used the command in write mode to set up an alternate display file.

LN Print screen. Copies current screen to 2040 printer.

ON NAMES AND NAMING

HOT Z's labelling or naming system is intended to make the programs you are reading or writing more comprehensible when they are listed. The four-letter limit is imposed by the 32-column display. A space is not a legel character in a HOT Z NAME, so use a dash or other punctuation if you want fewer than four letters. A semicolon is also illegal, since it is the escape character for the assembly editor.

The NAMEs themselves and the addresses they assigned to are contained in a special file, referred to as the NAME file. A NAME file is an ordered list beginning with the highest address to which a NAME is assigned (two bytes), then the four letters of that NAME, then the next highest address, etc. After the last NAME in a file, there must be two zero bytes. HOT Z takes care of ordering the NAMEs for you.

A small NAME file is loaded every time HOT Z is loaded, and that file contains four-letter abbreviations of the system variables as well as HOT Z's variables. You will find a few extras among the system variables. LINK, TADD, and ASIM are used by the single stepper. TEM1 through TEM9 are slots for temporary 16-bit variables for various HOT Z routines. (You may use them for any of your own routines for values that are not required once the routine is over, provided your routine does not call the floating-point calculator.)

The permanent NAME file that loads with HOT Z can be expanded to hold any NAMEs you add in a session of using HOT Z, or you have the option of starting a new file from scratch. In the standard version, the permanent NAME file is located just above a large work area, and as you add NAMEs the file expands *downwards* in memory (to lower addresses).

Add a NAME to the file with the INKEY$ command in WRITE mode with a disassembly (not data) on screen. The command will give you a cursor in the NAME column and allow you to enter or replace the NAME for that address. A legal NAME is made up of any four single characters with the restriction that at least one character must be beyond F in the alphabet. If you forget that rule, HOT Z will refuse to accept your new NAME and will ask you for another. A space in a NAME will be accepted and the disassembler will list the NAME, but you will not be able to use such NAMEs when working with the assembler, which parses according to spaces and punctuation. Take care that your NAMEs are unique, or HOT Z will always find only the one at the higher address when you refer to it. (If you enter a NAME to the ADDR cursor before you assign it, then the NAME file will be searched and the display will move to that NAME if it is already there; otherwise the display stays put.)

The EXP key (WRITE) will delete a NAME at the cursor address from the screen and from the NAME file.

The RND command (READ) is there to let you find the start of your current NAME file. You may want to check up on it if your are working under crowded memory conditions to be sure the file doesn't overwrite some valuable code. This command switches the display to data and moves to the lowest address of the NAME file. Since the NAME column in the data display lists NAMEs assigned to addresses formed by pairs of bytes in the hex column, the NAME appears horizontally across from the first address byte and then vertically opposite the last four data bytes. (Be aware that chance occurrences of data can look like addresses and cause spurious listings in the NAME column of the data display.)

You should also use the RND command when it comes time to SAVE the NAMEs you have entered in a session. However, you will also need to know the end address of your file in order to SAVE it. You can call up that end address by entering NEND to the ADDR cursor; the end address of the NAME file is listed lo-hi there. You can either add 2 to that address to include the two zero bytes that act as a terminator, or you can remember to zero those two bytes after you reload the tape. If you choose the first option, hit RND, turn on the edit cursor, set END to NEND+2, and SAVE. Record the addresses for use when you reload.

When you reload a NAME file, you must install the start and end addresses so that HOT Z will know where to look for that file. This is done at the four-byte block labelled ALNA (alternate NAMEs). With the data display and the edit mode, write the start address twice (lo-hi) followed by the NEND address; don't forget to subtract 2 if you have included the terminating zeroes. (If you have not included them, make sure they are there *first*.) If you don't do these settings correctly, you will hang up the program when you try to switch the new file on.

The NAME-file switch command is OVER in READ. It will switch from the permanent NAME file to the one you have loaded, after you have installed the file parameters at ALNA. If you use OVER without installing the new parameters, the effect will be to switch off the NAMEs entirely and you will not be able to add new ones. You should switch off the permanent NAME file in this way before loading a new file; then install the start and end addresses of the new file at ALNA and use OVER to switch them in.

A large NAME file is included on your master tape to give you some idea of the main ROM and HOT Z routines. Loading instructions are at the beginning of the notes. An annotation will be provided if there is sufficient demand for it.

If you save HOT Z using the BASIC routine on the master tape, then your current label file will be saved along with HOT Z and will load again from the same tape. Thus you can pick up on a given project without creating a separate NAME file.

You can amalgamate NAME files only if they pertain to separate blocks of memory, with the addresses in one block all higher than those in the other. Then just load the two files end to end in the proper order and save them as a single file.

SOME IMPORTANT HOT Z NAMES

AFEX	Store for AF' register pair in single-stepper
AFRG	Store for AF register pair in single-stepper
ALNA	Alternate NAME file descriptors. Six bytes.
ASIM	Single-step simulation area. Five bytes.
BCEX	Store for BC' register pair in single-stepper
BCRG	Store for BC register pair in single-stepper
BPT2	Breakpoint #1 address
BPT2	Breakpoint #2 address
CADR	Current address for disassembly
CBFL	Flag for a bit-op prefix (CB)
CHOO	Selects and updates Read mode display
COUN	Counter for printing register values
DEEX	Store for DE' register pair in single-stepper
DERG	Store for DE register pair in single-stepper
EDDQ	Flag for ED prefix
EOFA	The END address
FCBQ	Flag for prefixed bit ops
FENS	Single-step window switch; holds CRUN if off
FILC	Fill character, normally zero for screen clear
HLEX	Store for HL' register pair in single-stepper
HLRG	Store for HL register pair in single-stepper
IXRG	Store for IX register pair in single-stepper
IYRG	Store for IY register pair in single-stepper
KADD	Address pointed to by the cursor
KEYB	Gets code of keystroke into A; preserves other regs
KLIN	Line number with cursor
KPOS	Screen address of the cursor
KRED	Puts cursor address into HL and KADD
LENI	Length of current instruction in disassembly
LFPO	Stores address for floating-point interpreter
LOSI	Last one-step instruction
NADD	Next address for disassembly
NASW	Switch for NAME lookup
NEND	End of NAME list
NOSI	Next one-step instruction
NTOP	Most recent leading (low) address of NAME file
OSDF	One-step display file for extra window
OSDP	One-step display point for window, as DFCP
OVER	Overflow warning for User's stack
POIN	Pointer used in building register-value display
PRIM	Space or prime for register display
SPBI	Stack-pointer storage bin for stack switches
UNDR	Underflow warning for User's stack
USRS	Single-step user's stack pointer. Sets with S.

USING THE RELOCATE COMMANDS (MOVE, STR$, CHR$)

The Relocate command is rather complex in order to provide you a degree of flexibility in relocating your routines. A set of nine addresses must be entered before using the MOVE command, and a certain amount of planning and knowledge of the subject program is required to derive the correct addresses. Simple programs with one or two calls or absolute jumps are best labelled, moved with the Transfer-with-NAMEs (MERGE) command, and then fixed up by hand.

A program of reasonable complexity will have a block of code, a block of data (which may include address lists or jump tables), and a block of variables. Good programming form would recommend that you keep these blocks separate and distinct rather than, say, mingle data and variable storage in the crannies between your subroutines. If you are programming with HOT Z, you can separate the blocks generously as you develop your program and then use the Relocate command to close the gaps when you finish.

HOT Z's Relocate command will work on program blocks where code, data and variables are separate and distinct. If you have embedded patches of data, the command may still work, but you should check the data after the relocation to make sure that it has not been changed under the guise of readdressing code. Programs such as the 2Ø68 ROM, where jump tables lie around like empty beer cans, would have to be broken up into segments and relocated piecemeal.

The Relocate routine readdresses and moves Z8Ø code. However, the command does not take account of overlapping segments between source and destination blocks, so you cannot directly relocate a program to addresses already occupied by that program. (In such cases, you should use the transfer command first and then readdress in place with the relocate command.)

Jump tables have to be revised with the STR$ command, which first asks you for a displacement and then adds that displacement to each address in the file, starting at the cursor and ending at the END address. (If you moved your code from 81ØØH to 84ØØH then the displacement would be Ø3ØØH; from 84ØØH to 81ØØH would be a displacement of FDØØH.) Jump tables and data blocks should be moved with the Transfer command prior to using the relocate command.

The Relocate command (MOVE) allows you to move the code block by one displacement, the data block by another, and the variables block by a third displacement. (Any other three-way separation should also work.)

ADDRESS ENTRY FOR RELOCATING

The variables TEM1 through TEM9 are used to set the nine address parameters for relocation. The nine addresses are three sets of three addresses. Each set of three addresses indicates the start and end of an address range to be changed and the start address of the new address range. For example, suppose your program to be relocated fit the following memory map:

84D0-84E8	Variables
84F0-84FF	Data
8500-8680	Program

Suppose you want to put the variables and data at 8100H and the program at AC40. First, transfer the variables block to 8100H; it will run to 8118, so transfer the data block to 8119-8128. To move the program from 8500 up to AC40, any addresses of jumps or calls that lie between 8500 and 8680 should be changed to lie between AC40 and ADC0. (You don't need that last number.) So enter the original range in TEM1 and TEM2 and the first address of the new block in TEM3, thus:

TEM1	8500
TEM2	A680
TEM3	AC40

These first three TEM values always hold the parameters relating to the program (code) block. Variables and data parameters can go interchangeably into TEM4-TEM6 or TEM7-TEM9.

Addresses of variables, which were at 84D0-84E8, must be changed to start at 8100, and addresses of data, formerly at 84F0-84FF, must be changed to begin at 8119, so fill in the remaining TEM slots as follows:

Variables		Data	
TEM4	84D0	TEM7	84F0
TEM5	84E8	TEM8	84FF
TEM6	8100	TEM9	8119

TEM4-6 are one block, TEM7-9 the other. Now set the cursor at 8500 (start of the code segment) and set END to 8680, then give the MOVE command. The code will be copied to the new location and readdressed to run with the new variables, new data block, and any relocated subroutines in the code block. The original code will remain unchanged at its original location.

You may also use the Relocate command to split a code block into two or more separate blocks, but you must apply it repeatedly, once for each of the end-product blocks, and readdress for the blocks that are not being moved as if those blocks were variables or data.

If you lack variables or data blocks, then use a single non-zero dummy value for all three of the second or third set of TEM values, i.e., make them all three the same.

The relocator leaves unchanged any ROM calls or any loads to or from the systems variables area (5C00-6000).

After you have relocated a program, you may want to readdress a block of NAMEs that pertain to it. The command on the CHR$ key will do this for you. The CHR$ command works just like the STR$ command, except that it readdresses every third pair of bytes. Just enter the proper displacement. If you are readdressing only part of a label file, you may have to do some block moves to keep all the addresses in inverse sequence. Labels will be lost (from the screen, not the file) if you destroy the ordering of the addresses.

THE FLOATING-POINT INTERPRETER

RST 28H is the entry into the ROM's floating-point operations, which are coded in the bytes between an RST 28 and the following 38H. There is a good explanation of this second language (Or is it third?) of the ZX in Dr Logan's article in SYNC 2,2. (But beware of the two sign tests, which aren't jumps, as labelled in SYNC.) Note also that there have been a few changes for the 2068 ROM.

HOT Z will read this floating-point language, but only after you turn on the floating-point interpreter (PEEK in READ). If you leave the floating-point interpreter turned on, you will get a true reading of the ROM, but problems can arise elsewhere in memory when you encounter an EF that functions as data rather than an RST 28. You may get locked into the floating-point interpreter mode, without a 38H, the END character, in sight. The way out from this barrage of gibberish is the PEEK command again, which switches out the floating-point interpreter entirely. Other times you may want to read it, because this extra language is really one of the treats of the Sinclair-calculator heritage.

The f-p interpreter is also turned off by entry of a numerical address, but not by a page flip or a NAME, so use the last two when you're working with f-p. In addition, there is a special key command, CODE in READ mode, which switches the flag that tells the disassembler which language it's in.

The CODE command (READ) has a dual purpose. It will get you out of floating-point mode (without turning off the interpreter) if you need to and can't, or it will get you in when you want to be but aren't. You may get stuck in that mode through addressing yourself into the middle of a Z80 instruction, for example. Since floating-point operations include jumps and loops, there are also inclusions of f-p code that do not begin with an RST 28, branches of jumps. The CODE command will get you into those branches. However, the command is just a bit switch and it doesn't function when the screen page itself switches from one language at the top to the other at the bottom. The cure, when the CODE command doesn't function is the trick of hitting the THEN key twice. This picks up the language mode from the bottom of the page to the top and reverses the reading of any bytes from one language to the other.

You will also encounter some queer behavior if there is f-p code at the bottom of the screen and you try to write or go to the One-Step. This is not generally fatal and can be cured by going back to disassembly and setting the screen so that it ends in Z80 disassembly. If you want to write f-p code, the only manageable way is to go into EDIT mode in data.

Floating-point operations are FORTH-like stack manipulations and easy to follow if you know something about that language. They use the MEM area of the systems variables as storage slots for six floating-point numbers. (Each is five bytes.) The f-p operations that transfer between the calculator stack and MEM are called GET and STOR and are followed by a single digit from Ø to 5 to indicate the slot used. Numbers or letters higher than 5 generally indicate a patch of nonsense with GET, STOR and STAK as well.

Many of the possible f-p operators do not occur in the coding of the ROM, where you are likely to encounter them with HOT Z. They occur instead during the ROM's reading of BASIC programs, and they are generally identical with a BASIC instruction. You could learn to write floating-point code with these and the purely machine-code f-p operators if you wanted to; it would be similar to BASIC and a little faster. The 'entry point' of these BASIC f-p operators into the real machine world is through the operation labelled RAFP (Run A as Floating-Point). However, you need only use the command numbers listed as the first column of the instruction list to perform those BASIC functions on whatever floating-point numbers are on the calculator stack. From the perspective of a HOT Z user, RAFP would be used only to run an operation that resulted from some calculation, whose result was a code in A.

Two of the f-p operations deliver data directly from the code listing to the calculator stack. They generally do this in an efficient way, using fewer than five bytes, if possible, to encode the five-byte floating-point number. HOT Z prints the encoded floating-point number in the NAME and mnemonics columns of the disassembly listing. Since the interpreter doesn't know where any number will end, it is necessary to begin all of them slightly out of column, or the longest would run into the next line and mess up the display file. The f-p interpreter also reads the full five hex bytes that go onto the f-p stack, rather than the condensed version that actually occurs in the ROM. The ADDR column keeps accurate track, and you can work out the extra bytes, which are generally trailing zeroes, from that column.

HOT Z prints floating-point data by using the same ROM routines that handle that data, so the disassembly slows down and becomes jerky when it has to print those huge numbers, or their single-digit versions.

The two data-stacking operations are labelled STFP (stack floating point) and APPX (approximator) The first of these puts one five-byte number on the calculator stack, the second a series of one to 31 (whatever is left when you AND the low nibble of the instruction byte with ØF) five-byte f-p constants. (That's 5 to 155 bytes.) The approximator uses anything from six to a dozen floating point constants to get to a value for Chebyshev polynomials to approximate the transcendental BASIC functions.

FLOATING POINT OPERATIONS

Code	Op	Addr	Description
00	JRT	3AAA	Jumps if stack top holds a true
01	SWOP	37FB	Exchanges the top and second 5-byte stack entry
02	DROP	3760	Throws away top stack entry
03	SUB	33CE	Subtracts top stack from second stack entry
04	MULT	3489	Multiplies top two stack entries and leaves product on stack
05	DIV	356E	Divides second entry by top stack, leaves quotient on stack
06	PWR	3C6C	Raises 2nd on stack to power of stack top
07	OR	3936	Performs BASIC OR on two top stack entries and leaves result
08	AND	393F	Performs BASIC AND on two top stack entries, leaves result
09	N<=M	3956	Numeric inequality test
0A	N>=M	3956	Numeric inequality test
0B	N<>M	3956	Numeric inequality test
0C	N>M	3956	Numeric inequality test
0D	N<M	3956	Numeric inequality test
0E	N=M	3956	Numeric equality test
0F	ADD	33D3	Adds two top stack entries and leaves sum on stack
10	$AND	3948	ANDs a string with a number
11	$<=	3956	String inequality test
12	$>=	3956	String inequality test
13	$<>	3956	String inequality test
14	$>	3956	String inequality test
15	$<	3956	String inequality test
16	$=	3956	String equality test
17	$TR+	39B7	Concatenates strings addressed by the two top stack entries
18	VAL$	39F9	BASIC Function
19	USR$	38D7	BASIC Function
1A	RDIN	3A60	Read in data from channel in A
1B	NEG	382D	Changes the sign of top stack entry
1C	CODE	3A84	Replaces top stack entry with its sinclair code
1D	VAL	39F9	BASIC function
1E	LEN	3A8F	BASIC function
1F	SIN	3BD0	BASIC function
20	COS	3BC5	BASIC function
21	TAN	3BF5	BASIC function
22	ASN	3C4E	BASIC function
23	ACS	3C5E	BASIC function
24	ATN	3BFD	BASIC function
25	LN	3B2E	BASIC function
26	EXP	3ADF	BASIC function
27	INT	3ACA	BASIC function
28	SQRT	3C65	BASIC function
29	SGNM	3851	BASIC function
2A	ABS	3829	BASIC function
2B	PEEK	386B	BASIC function
2C	INX_	3864	BASIC function

2D	USR#	3872	BASIC function
2E	STR$	3A3A	BASIC function
2F	CHR$	39E4	BASIC function
30	NOT	391C	BASIC function
31	DUP	377F	Duplicates top of stack (5 bytes)
32	QREM	3ABB	Replaces number pair by quotient on stack top, remainder below
33	JRU	3AA1	Unconditional relative jump
34	STFP	3785	Composes and stacks number from following data bytes
35	LONZ	3A95	Loop jump as DJNZ with BERG as counter
36	N<00	3921	Tests sign of stack top and replaces with true if negative
37	N>00	3914	Tests sign of stack top and replaces with true if positive
38	END	3AB6	Ends an RST 28 routine
39	AADJ	3B9E	Adjusts angle values modulo 2 pi for trig functions
3A	ROUN	35D3	Rounds down to integer
3B	RAFP	3761	Runs byte in A as f-p op code for BASIC functions
3C	DEXP	310D	Decimal exponent processor
80	APPX	3808	Successive approximator; stacks and processes constants
A0	STAK	37DA	Stacks 0,1,0.5,PI/2,or 10, depending on second nibble
C0	STOR	37EC	Recalls stored entry from calculator MEM slot in 2nd nibble
E0	GET	37CE	Stores entry in calculator MEM slot given by 2nd nibble